

Sky130nm tutorial

Carsten Wulff, carsten@wulff.no

I. TOOLS

I would strongly recommend that you install all tools locally on your system.

For the analog toolchain we need some tools, and a process design kit (PDK).

- [Skywater 130nm PDK](#). I use [open_pdk](#) to install the PDK
- [Magic VLSI](#) for layout
- [ngspice](#) for simulation
- [netgen](#) for LVS
- [xschem](#)
- python > 3.10

The tools are not that big, but the PDK is huge, so you need to have about 50 GB disk space available.

A. Setup WSL (Applicable for Windows users)

Install a Linux distribution such as Ubuntu 24.04 LTS by running the following command in PowerShell on Windows and follow the instructions.

```
wsl --install -d Ubuntu-24.04
```

When you have installed the Linux distribution and signed into it, install make

```
sudo apt install make
```

B. Setup public key towards github

Do

```
ssh-keygen -t rsa
```

And press “enter” on most things, or if you’re paranoid, add a passphrase

Then

```
cat ~/.ssh/id_rsa.pub
```

And add the public key to your github account. Settings - SSH and GPG keys

C. Provide git with author identity

There are interactions with git that require an author identity. You are supposed to use one of these interactions a lot during the project, namely, `git commit`. What you need to provide is an email address and a name. If you would like to keep your real email address private/secret, read what it says on GitHub at your user settings page under [emails](#). Use the below commands to provide the author identity information to git.

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

D. Get AICEX and setup your shell

You don’t have to put `aicex` in `$HOME/pro`, but if you don’t know where to put it, chose that directory.

```
cd
mkdir pro
cd pro
git clone --recursive https://github.com/wulffern/aicex.git
```

You need to add the following to your `~/.bashrc` (note that `~` refers to your home directory `$HOME/`. `.bashrc` also works, or `$HOME/.bash_profile` on some newer macs)

```
export PDK_ROOT=/opt/pdk/share/pdk
export LD_LIBRARY_PATH=/opt/eda/lib
export PATH=/opt/eda/bin:$HOME/.local/bin:$PATH
```

E. On systems with python3 > 3.12

On newer systems it’s not trivial to install python packages because python is externally managed. As such, we need to install a python environment.

```
#- Find a package similar to name below
sudo apt-get update
sudo apt install python3.12-venv
sudo mkdir /opt
sudo mkdir /opt/eda
sudo mkdir /opt/eda/python3
sudo chown -R $USER:$USER /opt/eda/python3/
python3 -m venv /opt/eda/python3
```

Modify the `~/.bashrc` to include the python environment

```
export PATH=/opt/eda/bin:/opt/eda/python3/bin:$HOME/.local/bin:$PATH
```

F. Install Tools

Make sure you load the settings before you proceed

```
source ~/.bashrc
```

Hopefully the commands below work, if not, then try again, or try to understand what fails. There is no point in continuing if one command fails.

```
cd aicex/tests/
make requirements
make tt
```

On a mac, you probably need to add bison to the path

```
export PATH="/opt/homebrew/opt/bison/bin:$PATH"
```

I’ve split the install of each of the tools. It’s possible to run the commented out lines instead, but they often fail

```
#make eda_compile
#sudo make eda_install
make magic_compile magic_install
make netgen_compile netgen_install
make xschem_compile xschem_install
make iverilog_compile iverilog_install
make ngspice_compile # Sometimes fails
make ngspice_compile ngspice_install
```

On Mac, do

```
brew install yosys verilator
```

On Linux, do

```
make yosys_compile yosys_install
```

On all, do

```
python3 -m ensurepip --default-pip

python3 -m pip install matplotlib numpy click svgwrite \
    pyyaml pandas tabulate wheel setuptools tikzplotlib
source install_open_pdk.sh
```

G. Install cicconf

cicConf is used for configuration. How the IPs are connected, and what version of IPs to get.

```
cd
cd pro/aicex/ip/cicconf
git checkout main
git pull
python3 -m pip install -e .
cd ../
```

Update IPs

```
cicconf clone --https
cd ../../
```

H. Install cicsim

cicSim is used for simulation orchestration.

```
cd aicex/ip/cicsim
python3 -m pip install -e .
cd ../../
```

I. Setup your ngspice settings

Edit ~/.spiceinit and add

```
set ngbehavior=hsa      ; set compatibility for PDK libs
set ng_nomodcheck      ; don't check the model parameters
set num_threads=8      ; CPU hardware threads available
set skywaterpdk
option noint           ; don't print operating point data
option klu
optran 0 0 0 100p 2n 0 ; don't use dc operating point,
option opts
```

II. CHECK THAT MAGIC AND XSCHM WORKS

To check that magic and xschem works

```
cd ~/pro/aicex/ip/sun_sar9b_sky130nm/work
magic ../design/SUN_SAR9B_SKY130NM/SUNSAR_SAR9B_CV.mag &
xschem -b ../design/SUN_SAR9B_SKY130NM/SUNSAR_SAR9B_CV.sch &
```

III. DESIGN TUTORIAL

A. Create the IP

I've made some scripts to automatically generate the IP.

To see what files are generated, see tech_sky130A/cicconf/ip_template.yaml

```
cd aicex/ip
cicconf newip ex
```

B. The file structure

It matters how you name files, and store files. I would be surprised if you had a good method already, as such, I won't allow you to make your own folder structure and names for things. I also control the filenames and folder structure because there are many scripts to make your life easier (yes, really) that rely on an exact structure. Don't mess with it.

1) *Github workflows*: On github it's possible use something called workflows to run things every time you push a new version. It's really nice, since it can then check that your design is valid.

The grading of the milestones is determined by passing github workflows.

We will also check that you have not cheated, and modified the workflows just to get them passing.

The workflows are defined below.

```
.github
workflows
docs.yaml # Generate a github page
drc.yaml  # Run Design Rule Checks
gds.yaml  # Generate a GDS file from layout
lvs.yaml  # Run Layout Versus Schematic
          # and Layout Parasitic Extraction
sim.yaml  # Run a simulation
```

2) *Configuration files*: Each IP has a few files that define the setup, you'll need to modify at least the README.md and the info.yaml.

```
.gitignore # files that are ignored by git
README.md  # Frontpage documentation
config.yaml # What libraries are used. Used by cicconf
info.yaml  # Setup names, authors etc
media      # Where you should store images for documentation
tech -> ../tech_sky130A # The technology library
```

3) *Design files*: A "cell" in the open source EDA world should consist of the following files

- Schematic (.sch)
- Layout (.mag)
- Documentation (.md)

The files must have the same name, and must be stored in design/<LIB>/ as shown below.

Note there are also two symbolic links to other libraries. These two libraries contain standard cells and standard analog transistors (ATR) that you should be using.

```
design
JNW_EX_SKY130A
JNW_EX.sch
JNW_ATR_SKY130A -> ../../jnw_atr_sky130a/design/JNW_ATR_SKY130A
JNW_TR_SKY130A -> ../../jnw_tr_sky130a/design/JNW_TR_SKY130A
```

For example, if the cell name was JNW_EX, then you would have

- design/JNW_EX_SKY130A/JNW_EX.sch: Schematic (xschem)
- design/JNW_EX_SKY130A/JNW_EX.sym: Schematic (xschem)
- design/JNW_EX_SKY130A/JNW_EX.mag: Layout (Magic)
- design/JNW_EX_SKY130A/JNW_EX.md : Mark-down documentation (any text editor)

All these files are text files, so you can edit them in a text editor, but mostly you shouldn't (except for the Markdown)

4) *Simulations*: All simulations shall be stored in sim. Once you have a Schematic ready for simulation, then

```
cd sim
make cell CELL=JNW_EX
```

This will make a simulation folder for you. Repeat for all your cells.

```
sim
  Makefile
  cicsim.yaml -> ../tech/cicsim/cicsim.yaml
```

5) *The work:* All commands (except for simulation), shall be run in the `work` folder.

In the `work/` folder there are startup files for Xschem (`xschemrc`) and Magic (`.magirc`). They tell the tools where to find the process design kit, symbols, etc. At some point you probably need to learn those also, but I'd wait until you feel a bit more comfortable.

```
work
  .magirc
  Makefile
  mos.24bit.dstyle -> ../tech/magic/mos.24bit.dstyle
  mos.24bit.std.cmap -> ../tech/magic/mos.24bit.std.cmap
  xschemrc
```

C. Github setup

Create a repository on [github](#). The name of the repository that you make on GitHub has to be the same as what is written after `<your username>` in the last command below. In this example, that is `jnw_ex_sky130a`.

```
cd jnw_ex_sky130a
git remote add origin \
git@github.com:<your username>/jnw_ex_sky130a.git
```

D. Start working

1) *Edit README.md:* Open `README.md` in your favorite text editor and make necessary changes.

2) *Familiarize yourself with the Makefile and make:* I write all commands I do into a Makefile. There is nothing special with a Makefile, it's just what I choose to use 20 years ago. I'm not sure I'd choose something different now.

```
cd work
make
```

Take a look inside the file called `Makefile`.

E. Draw Schematic

The block we'll make is a current mirror with a 1 to 4 scaling.

A schematic is how we describe the connectivity, and the types of devices in an analog circuit. The open source schematic editor we will use is Xschem.

Open the schematic:

```
xschem -b ../design/JNW_EX_SKY130A/JNW_EX.sch &
```

1) *Add Ports:* Add `IBPS_5U` and `IBNS_20U` ports, the P and N in the name signifies what transistor the current comes from. So `IBPS` must go into a diode connected NMOS, and N will be our output, and go into a diode connected PMOS somewhere else.

2) *Add transistors:* Use 'I' or 'Shift+i' (note the letter case) to open the library manager. Click the `jnw_ex_sky130A/design` path, then `JNW_ATR_SKY130A` and select `JNWATR_NCH_4C5F0.sym`

The naming convention for these transistors is `<number of contacts on drain/source>C<times minimum gate length>F`, so the number before the C is the width, and the number before/after the F is the length. The absolute size does not matter for now. Just think "4C5F0 is a 4 contact wide long transistor", while a "4C1F2 is a 4 contact wide, short transistor".

Select the transistor and press 'c' to copy it, while dragging, press 'shift-f' to flip the transistor so our current mirror looks nice. 'shift-r' rotates the transistor, but we don't want that now.

Press ESC to deselect everything

Select the input transistor, and change the name to 'xi'

Select the output transistor, and change the name to 'xo[3:0]'. Using bus notation on the name will create 4 transistors

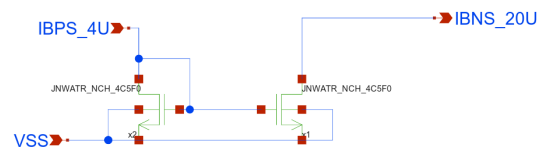
Select ports, and use 'm' to move the ports close to the transistors.

Press 'w' to route wires.

Use 'shift-z' and z, to zoom in and out

Use 'f' to zoom full screen

Remember to save the schematic



3) *Netlist schematic:* Check that the netlist looks OK

In work/

```
make xsch CELL=JNW_EX
cat xsch/JNW_EX.spice
```

F. Typical corner SPICE simulation

I've made `cicsim` that I use to run simulations (ngspice) and extract results

1) *Setup simulation environment:* Navigate to the `jnw_ex_sky130a/sim/` directory.

Make a new simulation folder

```
cicsim simcell JNW_EX_SKY130A JNW_EX \
../tech/cicsim/cell_spice/template.yaml
```

I would recommend you have a look at `simcell_template.yaml` file to understand what happens.

2) *Familiarize yourself with the simulation folder:* I've added quite a few options to cicsim, and it might be confusing. For reference, these are what the files are used for

File	Description
Makefile	Simulation commands
cicsim.yaml	Setup for cicsim
summary.yaml	Generate a README with simulation results
tran.meas	Measurement to be done after simulation
tran.py	Optional python script to run for each simulation
tran.spi	Transient testbench
tran.yaml	What measurements to summarize

The default setup should run, so

```
cd JNW_EX
make typical
```

3) *Modify default testbench (tran.spi):* Delete the VDD source

Add a current source of 5uA, and a voltage source of 1V to IBNS_20U

```
IBP 0 IBPS_5U dc 5u
V0 IBNS_20U 0 dc 1
```

Save the current in V0 by adding i(V0) to the save statement in the testbench

Save the voltage by adding v(IBPS_5U) to the save statement

```
.save i(V0) v(IBPS_5U)
```

4) *Modify measurements (tran.meas):* Add measurement of the current and VGS. It must be added between the "MEAS_START" and "MEAS_END" lines.

```
let ibn = -i(v0)
meas tran ibns_20u find ibn at=5n
meas tran vgs_m1 find v(ibps_5u) at=5n
```

Run simulation

```
make typical
```

and check that the output looks okish.

Try to run the simulation again

```
make typical
```

If everything works, then the simulation now should **not** be run. Every time cicsim runs (provided the sha: True option is set in cicsim.yaml) cicsim will compute a SHA hash of all files (stored in output_tran/.sha) that is referenced in the tran.spi. Next time cicsim is run, it checks the hash's and does not re-run if there is no need (no files changed).

Sometimes you want to force running, and you can do that by

```
make typical OPT="--no-sha"
```

Often, it's the measurement that I get wrong, so instead of rerunning simulation every time I've added a "--no-run" option to cicsim. For example

```
make typical OPT="--no-run"
```

will skip the simulation, and rerun only the measurement. This is why you should split the testbench and the measurement. Simulations can run for days, but measurement takes seconds.

5) *Modify result specification (tran.yaml):* Add the result specifications, for example

```
ibn:
  src:
    - ibns_20u
  name: Output current
  min: -20%
```

```
typ: 20
max: 20%
scale: 1e6
digits: 3
unit: uA
```

```
vgs:
  src:
    - vgs_m1
  name: Gate-Source voltage
  typ: 0.6
  min: 0.3
  max: 0.7
  scale: 1
  digits: 3
  unit: V
```

Re-run the measurement and result generation

```
make typical OPT="--no-run"
```

Open result/tran_Sch_typical.html

6) *Check waveforms:* You can either use ngspice, or you can use cicsim, or you can use something I don't know about

Open the raw file with

```
cicsim wave output_tran/tran_SchGtKttTtVt.raw
```

Load the results, and try to look at the plots. There might not be that much interesting happening

G. All corners SPICE simulations

Analog circuits must be simulated for all physical conditions, we call them corners. We must check high and low temperature, high and low voltage, all process corners, and device-to-device mismatch.

For the current mirror we don't need to vary voltage, since we don't have a VDD.

1) *Remove Vh and Vl corners (Makefile):* Open Makefile in your favorite text editor.

Change all instances of "Vt,Vl,Vh" and "Vl,Vh" to Vt

2) *Run all corners:* To simulate all corners do

```
make typical etc mc
```

where etc is extreme test condition and mc is monte-carlo.

Wait for simulations to complete.

3) *Get creative with python:* Open tran.py in your favorite editor, try to read and understand it.

The name parameter is the corner currently running, for example tran_SchGtAmcttTtVt.

The measured outputs from ngspice will be added to tran_SchGtAmcttTtVt.yaml

Delete the "return" line.

Add the following lines (they automatically plot the current and gate voltage)

```
import cicsim as cs
fname = name + ".png"
print(f"Saving {fname}")
cs.rawplot(name + ".raw", "time", "v(ibps_5u), i(v0)" \
, ptype="", fname=fname)
```

Re-run measurements to check the python code

```
make typical etc mc OPT="--no-run"
```

You'll see that cicsim writes all the png's. Check with `ls -l output_tran/*.png`.

You'll also notice it will slow down the simulation, so maybe remove the lines from `tran.py` again ;-)

4) *Generate simulation summary:* Run

```
make summary
```

Install [pandoc](#) if you don't have it

Run

```
pandoc -s -t slidy README.md -o README.html
```

to generate a HTML slideshow that you can open in browser. Open the HTML file.

5) *Viewing results without GUI browser:* If your on a system without a browser, or indeed a GUI, then it's possible to view the results in the terminal.

Check if `lynx` is installed, if it's not installed, then

On linux

```
sudo apt-get install lynx
```

On Mac

```
brew install lynx
```

Then

```
lynx README.html
```

6) *Think about the results:* From the corner and mismatch simulation, we can observe a few things.

- The typical value is not 20 uA. This is likely because we have a M2 VDS of 1 V, which is not the same as the VDS of M1. As such, the current will not be the same.
- The statistics from 30 corners show that when we add or subtract 3 standard deviation from the mean, the resulting current is outside our specification of $\pm 20\%$. I'll leave it up to you to fix it.

H. Draw Layout

A foundry (the factory that makes integrated circuits) needs to know how we want them to create our circuit. So we need to provide them with a "layout", the recipe, or instruction, for how to make the circuit. Although the layout contains the same components as the schematic, the layout contains the physical locations, and how to actually instruct the foundry on how to make the transistors we want.

Open Magic VLSI

```
cd work
magic ../design/JNW_EX_SKY130A/JNW_EX.mag
```

Now brace yourself, Magic VLSI was created in the 1980's. For it's time it was extremely modern, however, today it seems dated. However, it is free, so we use it.

1) *Magic VLSI:* Try google for most questions, and there are youtube videos that give an intro.

- [Magic Tutorial 1](#)
- [Magic Tutorial 2](#)
- [Magic Tutorial 3](#)

- [Magic command reference](#)
- [Magic Documentation](#)

Default magic start with the BOX tool. Mouse left-click to select bottom corner, left-click to select top corner.

Press "space" to select another tool (WIRING, NETLIST, PICK).

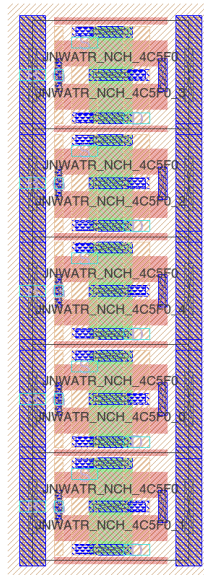
Type "macro help" in the command window to see all shortcuts

Hotkey	Function
v	View all
shift-z	zoom out
z	zoom in
x	look inside box (expand)
shift-x	don't look inside box (unexpand)
u	undo
d	delete
s	select
Shift-Up	Move cell up
Shift-Down	Move cell down
Shift-Left	Move cell left
Shift-Right	Move cell right

2) *Add transistors:* Open Cell -> Place Instance. Navigate to the right transistor.

Place it. Hover over the transistor and select it with 's'. Now comes a bit of tedious thing. Select again, and copy. It's possible to align the transistors on-top of each other, but it's a bit finicky.

Place all transistors on top of each other.



3) *Add Ground:* In the command window, type

```
see no *
see viali
see locali
see m1
see vial
see m2
```

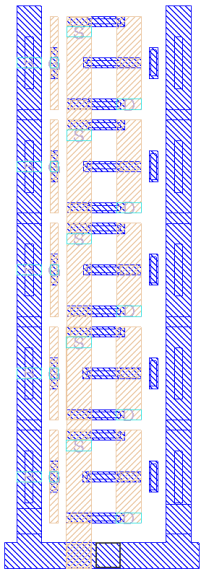
Change to the 'wire tool' with spacebar. Press the top transistor 'S' and draw all the way down to connect all of the transistors' source terminals.

Change grid to 0.5 um.

Select a 0.5 um box below the transistors and paint the rectangle with locali (middle click on locali)

Connect guard rings to ground. Use the 'wire tool'

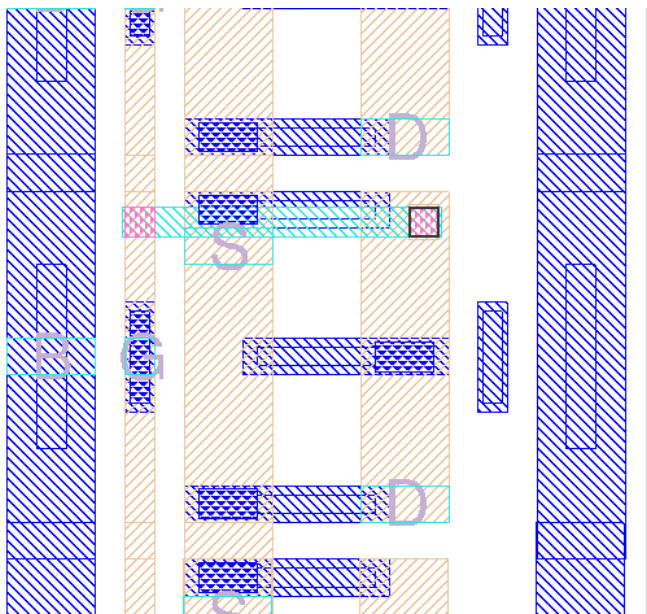
Connect the sources to ground. Use the 'wire tool'. Use 'shift-right click' to change layer down



4) *Route Gates:* Press "space" to enter wire mode. Left click to start a wire, and right click to end the wire.

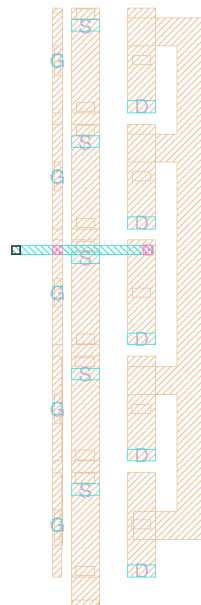
The drain of M1 transistor needs a connection from gate to drain. We do that for the middle transistor.

Start the route, press 'shift-left click' to go up one layer, route over to drain, and 'shift-right click' to go down.



5) *Drain of M2:* Use the wire tool to draw connections for the drains.

To add vias you can do "shift-left click" to move up a metal, and "shift-right click" to go down.



6) *Add labels:* Select a box on a metal, and use "Edit->Text" to add labels for the ports. Select the port button.

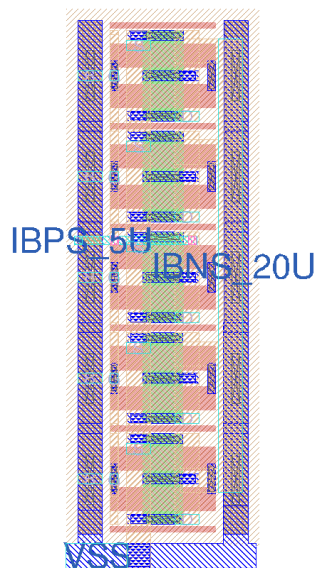
I. Layout verification

The DRC can be seen directly in Magic VLSI as you draw.

To check layout versus schematic navigate to work/ and do

```
make cdl lvs
```

If you've routed correctly, then the LVS should be correct.



J. Extract layout parasitics

With the layout complete, we can extract parasitic capacitance.

```
make lpe
```

Check the generated netlist

```
cat lpe/JNW_EX_lpe.spi
```

K. Simulate with layout parasitics

Navigate to `sim/JNW_EX`. We now want to simulate the layout. The default `tran.spi` should already have support for that.

Open the Makefile, and change

```
VIEW=Sch
```

to

```
VIEW=Lay
```

1) *Typical simulation:* Run

```
make typical
```

2) *Corners:* Navigate to `sim/JNW_EX`. Run all corners again

```
make all
```

3) *Simulation summary:* Open `summary.yaml` and add the layout files.

```
- name: Lay_typ
  src: results/tran_Lay_typical
  method: typical
- name: Lay_etc
  src: results/tran_Lay_etc
  method: minmax
- name: Lay_3std
  src: results/tran_Lay_mc
  method: 3std
```

Run summary again

```
make summary
pandoc -s -t slidy README.md -o README.html
```

Open the `README.html` and have a look at the results. The layout should be close to the schematic simulation.

L. Make documentation

Make a file (or it may exist) `design/JNW_EX_SKY130A/JNW_EX.md` and add some docs.

M. Edit info.yaml

Finally, let's setup the `info.yaml` so that all the github workflows run correctly.

Mine will look like this.

You need to setup the url (probably something like `<your username>.github.io`) to what is correct for you.

I've added the doc section such that the workflows will generate the docs.

The sim is to run a typical simulation.

```
library: JNW_EX_SKY130A
cell: JNW_EX
author: Carsten Wulff
github: wulffern
tagline: The answer is 42
email: carsten@wulff.no
url: analogicus.github.io
doc:
  libraries:
    JNW_EX_SKY130A:
      - JNW_EX
sim:
  JNW_EX: make typical
```

N. Setup github pages

Go to your GitHub repository (repo). Press Settings. Press Pages. Choose source under Build and Deployment -> GitHub Actions

Wait for the workflows to build. And check your github pages. Mine is https://analogicus.github.io/jnw_ex0_sky130a/.

O. Frequency asked questions

Q: My GDS/LVS/DRC action fails, even though it works locally.

Sometimes the reference to the transistors in the magic file might be wrong. Open the `.mag` file in a text editor and check. The correct way is

```
use JNWATR_NCH_4C5F0 JNWATR_NCH_4C5F0_0 ../JNW_ATR_SKY130A
```

It's the last `../JNW_ATR_SKY130A` that sometimes is missing.



Carsten Wulff received the M.Sc. and Ph.D. degrees in electrical engineering from the Department of Electronics and Telecommunication, Norwegian University of Science and Technology (NTNU), in 2002 and 2008, respectively. During his Ph.D. work at NTNU, he worked on open-loop sigma-

delta modulators and analog-to-digital converters in nanoscale CMOS technologies. In 2006-2007, he was a Visiting Researcher with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. Since 2008 he's been with Nordic Semiconductor in various roles, from analog designer, to Wireless Group Manager, to currently Principle IC Scientist. He's also an Adjunct Associate Professor at NTNU. His present research interests includes analog and mixed-signal CMOS design, design of high-efficiency analog-to-digital converters and low-power wireless transceivers. He is the developer of Custom IC Compiler, a general purpose integrated circuit compiler.